

Webアプリケーション入門 🚀

はじめに

はじめに：今日やること

- 今日のゴール: 
 - プログラミングがどういうものか知る
 - プログラミングの道具「VS Code」を試してみる
 - GitHubとGitHub Codespacesを試してみる
 - Linuxの基本的な操作を知る
 - 簡単なPythonプログラムを書いて動かす






身の回りのプログラム

- みんなが普段使っているスマホアプリ、ゲーム、YouTubeのようなWebサイト、お掃除ロボットやスマートスピーカーも、全部**プログラム**で動いています。



プログラミングって何？

- プログラミングとは、コンピュータに「こう動いてね」と順番に指示(命令)を書いて伝えること。
- その指示書が「プログラム」。
- 例：料理のレシピを書くようなもの。
 - 材料（データ）と手順（処理）を正確に書く必要がある。






コンピュータの仕組み

- コンピュータは大きく分けて：
 - i. 指示を**入力**する部分 (キーボード , マウス )
 - ii. 指示を記憶し**処理**する頭脳 (CPU , メモリ )
 - iii. 結果を**出力**する部分 (ディスプレイ )
- プログラムはこの**頭脳部分**で実行される。
- (補足) **OS** (Windows, macOS, Linuxなど) は、これらの部品やソフトウェア全体を管理するリーダーのような役割。




プログラミング言語とは？

- 人間：日本語、英語など  
- コンピュータ：機械語（0と1の世界）
- **プログラミング言語**は、人間が理解しやすい言葉で指示を書き、それをコンピュータがわかる言葉に翻訳してくれる「**通訳**」のようなもの。
- 今回は「**Python**」という、人間にとって読み書きしやすく、世界中で人気のある言語を使う。

プログラミングの流れ

1. プログラムを書く (VS Codeなどのエディタを使う) 
2. プログラムを実行する (コンピュータに指示を出す) 
3. 結果を確認する (画面に表示されたり、ファイルに保存されたり) 
4. エラーがあれば修正する (間違いを直す) 
5. 必要に応じて、プログラムを改良する (もっと便利にする) 

プログラミングを始めるために考えること

- 作ったプログラムをどこに保存する？ 
- どうやって書く？ 
- どうやって実行する？ 

それぞれ適材適所の道具がある！

- **VS Code:** プログラムを書くためのエディタ
- **GitHub:** プログラムを保存したり、みんなで協力して作るためのウェブサイト
- **GitHub Codespaces:** GitHub上でプログラムを書くための開発環境

VS Code とは？



- プログラムを書くための特別なメモ帳
- Microsoftが作ってる
- **無料**で使える！
- どのパソコンでもOK (Windows, Mac, Linux)

便利ポイント① コードが見やすい！ 🙄 ✨

「色分け」機能 (シンタックスハイライト)

```
# 例 : Pythonのコード
def greet(name):
    message = "こんにちは、" + name + "さん！"
    print(message)

greet("あなた")
```

- 命令や文字などを自動で色分け
- どこに何が書いてあるか、パッと見て分かりやすい！

便利ポイント② 入力が楽ちん！

「入力サポート」機能 (インテリセンス)

- 打ち間違いを減らせる
- 入力候補を出してくれる (スマホの予測変換みたい！)
- プログラムの命令を覚える手助けになる

便利ポイント③ 間違い探しを手伝う！ 🐛 🔍

「デバッグ」機能 🐛

- プログラムがうまく動かない… なぜ？
- VS Codeが問題箇所を見つける手伝いをしてくれる！
 - 一時停止したり 🛑
 - 中のデータを確認したり 👁

便利ポイント④ 変更記録を残せる！

「Git連携」機能 (バージョン管理)

- プログラムを書き直しても大丈夫！
- いつ、どこを変更したか**記録**しておける
- 「やっぱり元に戻したい！」も簡単 (ゲームのセーブポイントみたい！)

便利ポイント⑤ パワーアップできる！ 💪🔌

「拡張機能」

- スマホにアプリを入れる感覚！
- 見た目を変えたり、新しい機能を追加できる
- 自分が使うプログラミング言語をもっと便利にできる

なんでおすすめ？

- **無料**ですぐ始められる！
- 基本的な操作が**使いやすい**！
- 世界中で大人気 → **情報が多い** (困っても調べやすい)
- Webサイト、ツール、ゲーム… **色々な開発**に使える！

ファイル操作

- 新規ファイル: `Ctrl+N` / `Cmd+N` または エクスプローラーで右クリック
- ファイルを開く: `Ctrl+O` / `Cmd+O` または エクスプローラーでダブルクリック
- クイックオープン: `Ctrl+P` / `Cmd+P` + ファイル名入力
- 保存:
 - 上書き保存: `Ctrl+S` / `Cmd+S`
 - 名前を付けて保存: `Ctrl+Shift+S` / `Cmd+Shift+S`
- 閉じる: `Ctrl+W` / `Cmd+W` (タブの `x` ボタン)

エディタ基本操作 ①

- **テキスト編集:** 入力、削除、コピー(`ctrl+c`)、ペースト(`ctrl+v`)、カット(`ctrl+x`)
- **元に戻す (Undo):** `ctrl+z` / `cmd+z`
- **やり直す (Redo):** `ctrl+y` / `cmd+shift+z`
- **行の操作:**
 - 複製: `shift+alt+↑/↓` (Mac: `shift+option+↑/↓`)
 - 移動: `alt+↑/↓` (Mac: `option+↑/↓`)
 - 削除: `ctrl+shift+k` / `cmd+shift+k`

エディタ基本操作 ② ✨


- コメントアウト:
 - 行コメント: `Ctrl+/` (Mac: `Cmd+/`)
 - ブロックコメント: `Shift+Alt+A` (Mac: `Shift+Option+A`)
- インデント: `Tab` / `Shift+Tab`
- 検索/置換:
 - ファイル内: `Ctrl+F` / `Ctrl+H` (Mac: `Cmd+F` / `Cmd+Option+F`)
 - 全体検索: `Ctrl+Shift+F` (Mac: `Cmd+Shift+F`)


サイドバー (アクティビティバー)

左端のアイコンで主要機能へアクセス:

エクスプローラー (`Ctrl+Shift+E`): ファイル/フォルダ管理

検索 (`Ctrl+Shift+F`): プロジェクト全体検索

ソース管理 (`Ctrl+Shift+G`): Git連携 

実行とデバッグ (`Ctrl+Shift+D`): コード実行/デバッグ 

拡張機能 (`Ctrl+Shift+X`): 機能追加


コマンドパレット: 万能ツール

`Ctrl+Shift+P` (Mac: `Cmd+Shift+P`)

- VSCodeのほぼ全てのコマンドを検索・実行可能
 - 例: `>Format Document` (コード整形)
 - 例: `>Git: Commit` (Gitコミット)
 - 例: `>Preferences: Open Settings (UI)` (設定を開く)
- ファイル名入力でクイックオープン (`Ctrl+P` 相当) も可能
- 困ったらまずコマンドパレット！

統合ターミナル

`ctrl+@` または ``Ctrl+`` (バッククォート)


- VSCode内でターミナル (PowerShell, bash, zsh) を利用可能
- 複数のターミナルを開いたり、分割したりできる 
- Gitコマンドの実行、ビルド、スクリプト実行などに便利

設定のカスタマイズ ⚙️

`Ctrl+,` (Mac: `Cmd+,`) または コマンドパレット `>Preferences: Open Settings (UI)`

- ユーザー設定: VSCode全体に適用 👤
- ワークスペース設定: 現在のプロジェクトのみに適用 📁
(`.vscode/settings.json`)
- UIでの設定と、 `settings.json` (JSON形式) での直接編集が可能
- テーマ、フォント、インデント、拡張機能の挙動などをカスタマイズ

9. ウィンドウとエディタ分割

- 新しいウィンドウ: `Ctrl+Shift+N` / `Cmd+Shift+N`
- エディタ分割:
 - `Ctrl+\` (Mac: `Cmd+\`) で左右分割 
 - エディタ右上の分割アイコンをクリック
 - コマンドパレット `>Split Editor`
- タブ移動: `Ctrl+Tab`

GitHubを一言でいうと…

プログラムの設計図（ソースコード）を
インターネット上に保存したり、
他の人と協力して作るための

便利なウェブサイト

です！

GitHubでできること (主な便利ポイント 3つ)

1. プログラムの置き場所 (バックアップ)
2. 時間を戻せるタイムマシン機能 (バージョン管理)
3. みんなで協力しやすくする機能

機能① プログラムの置き場所 (リポジトリ) 🏠💾

- 作ったプログラムをネット上に保存！
 - プロジェクトごとに専用フォルダのように管理
 - これを **リポジトリ** と呼びます
- **メリット**
 - 自分のPCが壊れても**データが消えない！** (重要バックアップ)
 - ネットがあれば、いつでもどこでも見れる

機能② 時間を戻せるタイムマシン (バージョン管理)



- プログラムの変更履歴を自動で記録！
 - 「いつ」「誰が」「どこを」変えたか分かる
 - 変更記録を作ること **コミット** と言う
- **メリット**
 - 「あ！間違えた！」 → **簡単に元に戻せる！**
(まるでゲームのセーブポイント！)
 - 安心して色々試せる！

機能②補足: ブランチとマージ

- **ブランチ (Branch):**
 - 今のプログラムをコピーして、別の場所で安全に作業するイメージ
 - 「下書き」「実験用コピー」
- **マージ (Merge):**
 - 実験がうまくいったら、コピーした作業を元の流れに**合体**！
- **メリット:** 元のプログラムを壊さずに、新しい機能追加や修正に挑戦できる！

機能③ みんなで協力しやすくする機能 🤝

- チーム開発の強い味方！
- Pull Request (プルリクエスト):
 - 「こんな変更どうかな？」と相談&レビュー依頼
 - 良いコードかどうか、みんなで確認できる
- Issue (イシュー):
 - バグ報告 🐛、機能要望 ✨、タスク管理 ✅ など
 - プロジェクトの課題を共有する掲示板 📌

なんでGitHubを使うと良いの？(メリットまとめ)

- 安心！💪
 - 失敗しても元に戻せる (バージョン管理)
 - PCが壊れても大丈夫 (バックアップ)
- 学べる！📖
 - 他の人の上手なコードを見て勉強できる (オープンソース)
- 協力しやすい！🤝
 - チームでの開発がスムーズに進む
- 記録になる！📝
 - 自分の頑張りが見える形に (ポートフォリオ)

- GitHubは開発者の心強い味方！
- まずは自分のコードを保存してみよう！
- 使っていくうちに便利さが分かるはず！

プログラミングの準備って… 🤔

- 色々なソフトのインストールが必要…
- パソコンごとに設定が違う…
- エラーが出て進めない…
- **「環境構築」って難しい！**

そこで登場！ GitHub Codespaces

一言でいうと…

「インターネット上にある、あなた専用の開発スペース」のようなもの！

- GitHubが用意してくれる「開発用のパソコン」を借りるイメージ
- 面倒な準備はGitHubにおまかせ！

メリット①：難しい「環境構築」がいらない！

- プログラミングを始める前の面倒な準備は一切不要！
- ボタンひとつで、必要なものが揃った環境がすぐに使える。
- 環境設定で悩む時間がなくなり、すぐにコードを書き始められます。

メリット②：パソコンの性能を気にしない！

- 実際の作業はGitHubのパワフルなコンピュータ上で行われます。
- だから、自分のパソコンが古くても、パワーがなくても大丈夫！
- Webブラウザさえ動けば、タブレットからでも開発できちゃいます。
- いつでも、どこでも開発OK！ 🌐🏠

メリット③：自分のパソコンが汚れない！

- プロジェクトごとに、独立した開発スペースを使います。
- 自分のパソコンに色々な開発ツールを入れる必要がありません。
- **パソコンの中はいつもスッキリ綺麗！** 他のソフトとの衝突も心配なし。

プログラミング学習の「最初の壁」を低くして、コードを書くことに集中できる！ 💪



どうやって使うの？ 🤔

1. GitHubのプロジェクトを選ぶ
2. 「Code」 ボタン ▶️ 「Codespaceを作成」 をクリック
3. Webブラウザに開発画面が出る！

これだけで、プログラミングを始められる環境が手に入ります。

特別なソフトのインストールは不要！ 👍

STEP 1: リポジトリページへ移動

目的: Codespacesを作成したいGitHubリポジトリを開きます。

操作:

ブラウザで対象のGitHubリポジトリページにアクセスしてください。

例: `https://github.com/[ユーザー名]/[リポジトリ名]`

STEP 2: 「Code」 ボタンをクリック

目的: Codespaces関連のメニューを表示します。

操作:

リポジトリのファイル一覧の上にある、緑色の `< > Code` ボタンをクリックします。

STEP 3: 「Codespaces」 タブを選択

目的: Codespaceの作成・管理画面に切り替えます。

操作:

`Code` ボタンをクリックして表示されたドロップダウンメニューから、
「Codespaces」 タブを選択します。

- **Local:** ローカルにクローンする手順
- **Codespaces:** Codespaceを作成・管理するタブ

STEP 4: Codespaceを作成

目的: 新しいCodespaceの作成を開始します。

操作:

「**Create codespace on [ブランチ名]**」 ボタンをクリックします。

- デフォルトでは `main` ブランチが選択されていますが、▼から他のブランチを選択することも可能です。
- 既存のCodespaceがあれば、この画面に一覧表示され、再開できます。

STEP 5: 環境構築と起動

しばらく待機... 

- GitHubがバックグラウンドで処理を開始します。
- リポジトリ内の `.devcontainer/devcontainer.json` が読み込まれ、定義された開発環境（コンテナイメージ、拡張機能など）が構築されます。
- 初回や設定が複雑な場合は少し時間がかかります。

Linuxってなに？ (1/2) 🐧

ズバリ言うと…

コンピューターの基本ソフト(OS)の一種です！

- みんながよく知ってる **Windows** や **macOS** の仲間。



Linuxってなに？ (2/2)

Linuxの大きな特徴は…

1. 無料（タダ！）で使えることが多い

- Windowsみたいにお金がかからない！



2. オープンソース

- ソフトウェアの「設計図」が公開されている！
- 世界中の人が改良に参加できる。
- 透明性が高く、安心感がある。

なぜ開発初心者にLinux？

Linuxを使うと、開発の学習にこんないいことがあります！

メリット①：開発ツールが揃えやすい！

- プログラミング言語（Python, Ruby, C など）
- 開発必須ツール（Git , Docker  など）
これらのインストールが簡単なことが多い！
- 「パッケージ管理」という仕組みで、コマンド数行で導入完了！

例: `sudo apt update && sudo apt install git`


メリット②：実際の開発現場に近い環境

- Webサービスやアプリの裏側（サーバー）は、**Linux**で動いていることが非常に多い！
- 開発の初期段階からLinuxに触れることで、将来役立つスキルが身につく。
- 本番環境に近いところで開発できる。


メリット③：「黒い画面」と友達になろう！

- Linuxでは「**ターミナル**」（黒い画面に文字を打つやつ）をよく使います。
- 最初は難しそう？でも、これは開発者にとって**超重要スキル**！
 - 作業が早くなる ⚡
 - 自動化できる 🤖
 - サーバー管理に必須 ☁
- Linuxはこの**コマンド操作**を学ぶのに最適！

メリット④：無料 & 古いPCでもOK！

- OS自体が**無料**なので、開発環境のコストを抑えられる！ 
- Windowsより**動作が軽い**ことが多い。
- ちょっと古くなったパソコンでも、Linuxを入れて開発マシンとして復活させられるかも！

メリット⑤：学びやすく、情報も豊富

- OSの仕組みに触れる機会が多く、コンピューターの理解が深まる。
- 世界中に**ユーザーが多い**！ 
- 困ったことがあっても、Webで検索すれば**情報がたくさん**見つかる（コミュニティも活発）。

Linuxの操作は、多くの場合**ターミナル**（黒い画面）で**コマンド**を入力して行います。



最初は難しく感じるかもしれませんが、基本的なコマンドを覚えれば、ファイル操作などが効率的に行えるようになります。

最初に覚えるべき基本的なコマンドを紹介します。

ファイルシステム操作 (1/3)

`pwd` (Print Working Directory)

- 現在のディレクトリ（カレントディレクトリ）のフルパスを表示します。
- 自分が今どこにいるか確認する時に使います。

```
$ pwd  
/home/username/Documents
```

ファイルシステム操作 (2/3)

`ls` (List)

- ディレクトリ内のファイルやディレクトリの一覧を表示します。

```
$ ls  
file1.txt  mydir  notes.md
```


よく使うオプション:

- `ls -l`: 詳細表示 (パーミッション、所有者、サイズ、日時など)
- `ls -a`: 隠しファイル (`.` で始まるファイル) も表示
- `ls -al`: 詳細表示 + 隠しファイル表示

ファイルシステム操作 (3/3)

`cd` (Change Directory)

- ディレクトリを移動します。

```
cd Documents    # Documents ディレクトリへ
cd ..           # 一つ上のディレクトリへ 
cd ~            # ホームディレクトリへ
cd /            # ルートディレクトリへ
```

`mkdir` (Make Directory): ディレクトリを作成

```
mkdir new_folder
```

`rmdir` (Remove Directory): **空の**ディレクトリを削除 (空のみ)

```
rmdir old_folder
```

ファイル操作 (1/4)

touch

- 空のファイルを作成したり、ファイルのタイムスタンプを更新します。

```
touch new_file.txt
```

cat (Concatenate)

- ファイルの内容を**すべて**表示します。短いファイル向き。

```
cat memo.txt
```


ファイル操作 (2/4)

less

- ファイルの内容を1画面ずつ表示します。長いファイル向き。
- 操作:
 - `スペース` : 次ページ
 - `b` : 前ページ
 - `/キーワード` : 検索
 - `q` : 終了

```
less long_log_file.log
```

ファイル操作 (3/4)

`cp` (Copy)

- ファイルやディレクトリをコピーします。

```
# ファイルをコピー  
cp source.txt destination.txt  
  
# ディレクトリをコピー (中身ごと)  
cp -r my_directory backup_directory
```

- ディレクトリのコピーには `-r` オプションが必要です。

ファイル操作 (4/4)

mv (Move)

- ファイルやディレクトリを移動、または名前を変更します。

```
# 名前変更
mv old_name.txt new_name.txt

# 移動
mv my_file.txt /tmp/
mv my_folder ../other_folder/
```

rm (Remove)

- ファイルやディレクトリを削除します。 **注意: 元に戻せません！**

```
rm old_file.txt          # ファイル削除
rm -r old_directory      # ディレクトリ削除 (確認あり)
rm -rf data_folder       # ディレクトリ強制削除 (超危険!)
```

情報表示・ヘルプ ?

man <コマンド名> : コマンドのマニュアルを表示

```
man ls
```

<コマンド名> **--help** : 簡単なヘルプを表示

```
cp --help
```

clear : 画面の表示をクリア

```
clear
```

その他

`exit`

- 現在のターミナルセッションを終了します。
- ターミナルウィンドウを閉じるのと同じ効果です。

```
exit
```

学習のポイントと注意点

- **実際に試す:** どんどんコマンドを打ってみましょう！(安全な環境で)
- `man` や `--help` を活用: わからないことはすぐ調べる癖をつけましょう。
- **オプションを覚える:** `-l`, `-a`, `-r` など、よく使うものから覚えましょう。
- **大文字・小文字:** Linuxは区別します。正確に入力しましょう。A ≠ a
- `rm` は慎重に: 特に `rm -rf` はファイル/ディレクトリを**確認なしで完全に削除**します。実行前に必ず確認しましょう。

安全な環境で練習しましょう！

Linuxの基本コマンドまとめ 🐧

- `pwd` , `ls` , `cd` : 移動と確認
- `mkdir` , `touch` , `cp` , `mv` , `rm` : 作成、コピー、移動、削除
- `cat` , `less` : ファイル内容表示
- `man` , `--help` , `clear` , `exit` : ヘルプ、情報表示など

Pythonってなに？ ①

コンピューターと話すための「言葉」

- コンピューターに「こう動いてね！」とお願いするための言葉（プログラミング言語）の一つです。
- 人間が日本語や英語を使うように、コンピューターと対話するために使います。



Pythonってなに？ ②

特徴1: 読み書きしやすい！

- まるで英語の文章みたいに、自然に読めるコードが書けます。
- 文法のルールが比較的シンプル！
- 他の人が書いたコードも読みやすい。

```
# 例：画面に「こんにちは！」と表示する  
print("こんにちは！")
```

➡ シンプルで分かりやすい！

Pythonってなに？ ③

特徴2: すぐに試せる！

- 書いたコードをすぐに実行して、結果を確認できます。
 - 例えるなら、料理中に味見する感じ！
- 間違いを見つけやすく、修正しやすい。

➡ 初心者でも安心して始められる！

Pythonってなに？ ④

特徴3: 便利な道具箱（ライブラリ）がたくさん！

- 「ライブラリ」＝便利な機能が詰まった道具箱
- Webサイト用、データ分析用、AI用など、色々な道具箱があります。
- これを使えば、難しいことも短いコードで実現可能！

➡ 効率的に開発できる！

Pythonってなに？ ⑤

特徴4: どこでも使える！

- Windows
- Mac
- Linux
- などなど…

普段使っているパソコンで、すぐに開発を始められます！

➡ 環境を選ばない！

なんでPython？ ①

理由1: とにかく学びやすい！

- 他の言語に比べて、覚えるルールが少なめ。
- シンプルで直感的なので、プログラミングが初めてでも大丈夫！

なんでPython？ ②

理由2: 「できた！」を体験しやすい！

- 簡単なことから始められて、すぐに結果が見える！
 - 例: 「こんにちは！」と表示する
 - 簡単な計算をする
- 「プログラミングって面白い！」と感じやすい。

なんでPython？ ③

理由3: できることの幅が広い！

- Webサイト作り (ブログ、ツール)
- データ分析 (グラフ作成、予測)
- AI・機械学習 (画像認識、文章生成)
- 自動化 (ファイル整理、情報収集)
- その他いろいろ！

なんでPython？ ④

理由4: 仲間がたくさん！

- 世界中にPythonを使っている人がたくさんいます。
- インターネット上に情報が豊富！
 - 解説記事、動画
 - 質問できるコミュニティ
- 困ったことがあっても、解決策を見つけやすい。


- Pythonは**分かりやすく、学びやすい**！
- **すぐに試せて**、「できた！」を体験しやすい！
- Web、AI、自動化など、**色々なことができる**！
- **仲間がたくさんいて、情報も豊富**！

 **とても優しくてパワフルな言語です！**


最初のプログラム：Hello, World!

- `hello.py` ファイルに以下を入力してみよう。

```
print("Hello, World!")
```

- `print()` : カッコの中身を画面に表示する命令。
- `"Hello, World!"` : 表示したい文字。文字は `"` (ダブルクォーテーション) で囲む。
- 保存: 
 - コードを書いたら必ず保存！ (Ctrl+S または Cmd+S)
 - ファイル名の横の `●` が `×` に変わる。

プログラムの実行 ▶

- プログラムを実行する方法：
 - **方法1:** 右上の再生ボタン (▶ Run Python File) をクリック。 ▶
 - **方法2:** 下のターミナルに `python hello.py` と入力してEnterキー。 
 - **実行結果:**
 - ターミナルに `Hello, World!` と表示されれば成功！
 - コンピュータに指示を出して、実行させることができた！

文字や数字を表示してみよう

- `print()` を使って、自分の名前や年齢、メッセージを表示させてみよう。

```
print("鈴木 一郎") # 自分の名前など
print(16)           # 自分の年齢など
print("プログラミングは楽しい!")
```


- 文字は `"` で囲む。数字はそのまま書ける。
- **実行結果:**
 - 一行ずつ実行されて、書いたものが表示されるか確認しよう。

計算させてみよう $+$ $-$ \times \div

- Pythonで計算もできる。記号（演算子）を使ってみよう。 

```
print(1 + 2)      # 足し算
print(10 - 3)     # 引き算
print(4 * 5)      # 掛け算
print(20 / 4)     # 割り算
print(10 % 3)     # 割り算の余り

# 文字と数字を混ぜて表示
print("今日の点数は", 100, "点")
```

- `+`, `-`, `*`, `/`, `%` などの記号で計算できる。
- 文字と数字は `,` (カンマ) で区切って一緒に表示できる。
- 実行結果: 
 - 計算結果が正しく表示されているか確認しよう。

(発展) 変数：データを入れておく箱

- プログラムの中でデータ（文字や数字）を一時的に保存しておく場所が「**変数**」。
- 箱に名前をつけて、そこにデータを入れるイメージ。

```
name = "田中"    # nameという変数に "田中" を入れる
age = 17         # ageという変数に 17 を入れる

print("僕の名前は", name, "です。")
print("年齢は", age, "歳です。")

# 変数を使って計算もできる
print("来年は", age + 1, "歳になります。")
```

- `=` は「右の値を左の変数に入れる」という意味（**代入**）。
- 一度変数に入れておけば、後で何度も使える。


(発展) 入力：ユーザーと対話する

- `input()` を使うと、プログラム実行中にユーザー（使う人）にキーボードから文字を入力してもらうことができる。

```
print("あなたの名前を教えてください。")
my_name = input() # 入力された文字が my_name に入る

print(my_name, "さん、こんにちは!")

# 質問と入力を同時に行う
food = input("好きな食べ物は何ですか? 🍕🍔🍜")
print(food, "、おいしいですね!")
```

- `input()` で受け取ったデータは「変数」に入れて使う。
- 実行するとターミナルで入力待ちになるので、何か入力してEnterキーを押してみよう。 

エラーが出たら？

- プログラムは少しでも間違えるように動かない。😵
- 赤い文字で**エラーメッセージ**が出たら、それは「ここが違うかも？」というコンピュータからのヒント。
- よくある間違い:
 - `print` を `pront` と打ち間違える (スペルミス)
 - 文字を囲む `"` を忘れる、または全角 `"` で入力している
 - カッコ `()` の閉じ忘れ
- メッセージをよく読んで、どこが違うか探してみよう。
- エラーは怖くない！直せば動く！

AIってデバッグの助っ人？

- プログラミングのエラー探し（デバッグ）は大変… 🤯
- でも、**生成AI**がお手伝いしてくれる！
- AIは「エラー解決」や「コード理解」の**ヒント**をくれる
- ⚠️ **注意:** AIも間違えることがある！ 最後は**自分で確認**しよう

AIへの質問方法①：エラーメッセージの意味を聞く

エラーが出たら、AIにコピペして聞いてみよう！

聞き方例:

「`SyntaxError: invalid syntax` ってどういう意味？ どう直せばいい？」

「このエラーメッセージ（エラーメッセージを貼り付け）が出たんだけど、初心者にも分かるように原因を教えてください。」

ポイント: エラーメッセージ全体を正確に伝えよう

AIへの質問方法②：コードの意味を聞く 🤔

分からないコードがあったら、AIに聞いてみよう！

聞き方例:

「このコード（コードを貼り付け）は何をしてるの？ 簡単な言葉で教えて！」

「この `if` 文の条件って、どういう意味？」

ポイント: どの部分が分からないか具体的に示そう

AIへの質問方法③：バグ（間違い）の場所を聞く

うまく動かないコード… AIに原因を探ってもらおう！

聞き方例:

「〇〇になるはずが△△になります。どこがおかしい？（コードを貼り付け）」

「ボタンをクリックしても何も起こらないんだけど、原因は何だと思う？」

ポイント: 「どうなってほしいか」「実際どうなったか」を詳しく伝えよう

AIへの質問方法④：テストコードを作ってもらおう

自分のコードがちゃんと動くかチェックしたい！

聞き方例:

「この関数のテストコードを作ってくれる？（関数コードを貼り付け）」

ポイント: バグを早めに見つけるためにテストは大事！ AIに手伝ってもらおうと楽ちん


AIへの質問方法⑤：もっと良い書き方を聞く

動くけど、もっとスッキリさせたい！

聞き方例:

「このコード、もっとシンプルに書き直せない？（コードを貼り付け）」

「このコード、もっと初心者にも読みやすいように改善するならどうすればいい？」

ポイント: より良いコードを学ぶチャンス！ 

超重要！ AIを使うときの注意点 ①

AIの言うことを100%信じない！

- 提案されたコードは、必ず自分で読んで理解しよう
- 少しずつ試しながら適用しよう
- いきなり全部コピペは危険！

超重要！ AIを使うときの注意点 ②

個人情報や会社の秘密は絶対に入力しない！

- 名前、住所、パスワード、APIキー、秘密のコードなどは **NG**！ 🙅
- 見せたいコードは、大事な部分を隠すか書き換えて！ (例: `xxx` にする)

超重要！ AIを使うときの注意点 ③

状況をできるだけ詳しく伝える！

- 何を使って (例: Python , Linux)
- 何をしようとして
- どうなってほしいか
- 実際はどうなったか
- どんなエラーメッセージが出たか

詳しく伝えるほど、的確なアドバイスがもらいやすい！

- 生成AIはデバッグの強い味方！
- 使い方と注意点を守って、上手に付き合おう！
- プログラミング学習をもっと楽しく、効率的に！


GitHub Copilot とは？ & 準備

- **GitHub Copilot:** AIによるコーディング支援ツール (コード補完・生成)
- **利用に必要なもの:**
 - i. GitHub アカウント
 - ii. Copilot サブスクリプション (有料プラン / トライアル)
 - iii. VSCode 拡張機能のインストール:
 - `GitHub Copilot`
 - `GitHub Copilot Chat` (推奨)
 - iv. VSCodeでGitHubアカウント認証

Copilot: インラインサジェスション

- コーディング中に、AIが文脈を読んでコード候補をグレー表示
- 機能:
 - 単純な補完 (変数名、関数名など)
 - 行単位、複数行のコードブロック生成
 - 定型的なコード (ボイラープレート) の生成

Copilot: 候補の操作

- 受け入れ (Accept): `Tab` キー
- 拒否 (Reject): `Esc` キー or そのまま入力継続
- 他の候補:
 - 次へ: `Alt +]` (Mac: `Option +]`)
 - 前へ: `Alt + [` (Mac: `Option + [`)
- 複数候補パネル表示: `Ctrl + Enter` (Mac: `Cmd + Enter`) 
 - サイドパネルで候補を比較・選択できる

Copilot: コメントからコード生成

1. やりたいことを自然言語でコメントに書く 📝

```
// 配列を受け取り、各要素を2倍にして新しい配列を返す関数
```

```
# 与えられた数値のリストを2倍にする関数
```

2. コメントの次の行にカーソルを置く

3. Copilotがコードを提案 → `Tab` で受け入れ ✅

➡ 具体的で明確なコメントが精度向上のカギ！

Copilot Chat: 対話でコーディング支援

GitHub Copilot Chat 拡張機能が必要

- **チャットビュー:** アクティビティバーのアイコン / コマンドパレット `>Chat:`
`Open Chat View`
- **インラインチャット:** エディタ内で `Ctrl+I` (Mac: `Cmd+I`)
- 自然言語で質問や指示が可能
 - コード生成依頼
 - コードの説明
 - バグ修正の提案
 - テストコード作成 など

Copilot Chat: 主な使い方 (スラッシュコマンド) /

チャット内で / を入力すると便利なコマンドが使える:

- @workspace /explain : 選択範囲やワークスペースのコードについて説明を求める
- @workspace /fix : 選択範囲の問題点を修正するコードを提案
- @workspace /doc : 選択範囲のコードのドキュメントコメントを作成
- /help : 利用可能なコマンド一覧を表示

@workspace を付けると、より広いコンテキスト (開いているファイルなど) を考慮してくれます。

Copilot 利用の Tips & 注意点

- **Tips:**

- 明確なコメント、明確な指示を心がける
- 小さな関数、明確な変数名を意識すると Copilot も理解しやすい
- ステータスバーのアイコンで一時的にON/OFF可能

- **注意点:**

- **生成コードは必ず確認・検証する！** (バグ、非効率、脆弱性)
- 機密情報を含むコード/コメントは慎重に扱う
- コードのライセンスを確認する

Copilot は "副操縦士"、最終的な責任は開発者に

まとめ

- VSCodeは開発者にとって強力なツール
- GitHubは開発者のためのプラットフォーム
- Linuxは開発環境として最適
- Pythonは学びやすく、幅広い用途に使える言語
- AIが提案するコードを参考にしつつ、自分のスタイルを大切に

ご清聴ありがとうございました！